Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:
Kot Jarosław
Prusicki Jakub

Łamacz haseł

Projekt zespołowy
na studiach stacjonarnych
o kierunku **Informatyka**
Stopień **II**

Opiekun projektu:
dr inż. Paweł Paduch

Kielce, 2023

SPIS TREŚCI

1. Wstęp teoretyczny. 3
1.1. Algorytm Brute Force. 3
1.2. Metoda słownikowa. 4
2. Przykładowe rozwiązania. 7

1. Wstęp teoretyczny.

1.1. Algorytm Brute Force.

Algorytm siłowy (z ang. **brute force**) jest to taki algorytm, którego działanie można streścić następująco: polega on na sukcesywnym sprawdzaniu wszystkich możliwych kombinacji, w poszukiwaniu rozwiązania problemu. Dla niektórych klas problemów, jest to podejście, pod pewnymi warunkami, jedyne niezawodne. Stosując takie podejście w programowaniu, rozwiązujemy problem przez weryfikację i ocenę wszystkich wariantów postępowania. Gdybyśmy dysponowali, dowolnie długim czasem na poszukiwanie rozwiązania oraz zasobami sprzętowymi (na przykład nieskończenie dużą pamięcią operacyjną lub przestrzenią dyskową), to algorytm taki zakończy się zawsze powodzeniem. O związkach metody brute force, z kombinatoryką wspomniano nie bez przyczyny. Korzystając z kombinatoryki, można policzyć ile jest możliwych takich haseł. Po obliczeniach można stwierdzić, że wartość tego zagadnienia jest sumą wariacji z powtórzeniami. Dla przypomnienia, wariację z powtórzeniami wykorzystujemy wtedy, gdy chcemy wiedzieć ile możemy stworzyć różnych układów k-elementowych, mając do dyspozycji n-elementów, przy czym kolejność elementów w układzie jest istotna, a elementy mogą się powtarzać[1].

Niech:

$$n - \text{ilość liter w słowniku},$$

$$k - \text{ilość znaków w haśle},$$

$$N - \text{ilość możliwych haseł}$$

$$N = \sum_{k=1}^{k=n} W_n^k = W_n^1 + W_n^2 + \ldots + W_n^{n-1} + W_n^n =$$

$$= n^1 + n^2 + \ldots + n^{n-1} + n^n$$

Rys. 1.1.1 Wzór na liczbę kombinacji

Innymi słowy metoda siłowa rozwiązania jakiegoś zadania polega na wyczerpaniu wszystkich możliwości. Dla anagramów oznacza to wygenerowanie listy wszystkich możliwych łańcuchów ze znaków łańcucha s1 i sprawdzenie, czy s2 znajduje się na tej liście. Nie jest to jednak zalecane podejście, przynajmniej w tym przypadku. Zauważmy mianowicie, że dla ciągu znaków s1 o długości n mamy n wyborów pierwszego znaku, (n-1) możliwości dla znaku na drugiej pozycji, (n-2) na trzeciej pozycji itd. Musimy zatem wygenerować n! łańcuchów znaków. Dla ustalenia uwagi przyjmijmy, że *s1* składa się z 20 znaków. Oznacza to konieczność wygenerowania

```
In [3]:
import math
math.factorial(20)

Out[3]:
2432902008176640000
```

Rys. 1.1.2 Ilość możliwych łańcuchów.

łańcuchów znaków, a następnie odszukanie wśród nich ciągu s2. Można było dostrzec takie podejście już wcześniej - szczególnie,jeśli chodzi o to ile czasu zajmuje algorytm klasy O(n!), dlatego nie jest to polecane podejście do zagadnienia anagramów[2].

1.2. Metoda słownikowa

Ataki słownikowe niezmiennie pozostają jedną z najskuteczniejszych metod odzyskiwania haseł. Tam, gdzie próby siłowe nie są w stanie w rozsądnym czasie doprowadzić do jakichkolwiek rezultatów, dobry słownik może zdziałać cuda. Do skutecznego działania niezbędne jest odpowiednie oprogramowanie oraz właśnie dobry słownik. O ile programy takie jak John The Ripper czy hashcat są dobrze znane i łatwo dostępne, o tyle znalezienie dobrego słownika nie jest już wcale takie proste. W celu odzyskania haseł z odpowiadających im skrótów (hashy) możemy skorzystać z odpowiedniego oprogramowania. Możemy też odwiedzić jeden z wyspecjalizowanych serwisów internetowych. Często są one stanie w ciągu kilku sekund odzyskać hasła ze skrótów wyznaczonych za pomocą algorytmów takich jak: LM, NTLM, md2,sha224, ripeMD160, whirlpool, MySQL 4.1+. Do wykonania powyższego zadania serwisy wykorzystują ogromne tablice skrótów i odpowiadających im haseł o wielkościach liczonych w setkach gigabajtów. Odpowiednie tablicowanie wpływa na szybkość całego procesu, zaś o skuteczności decyduje jakość oraz rozmiar bazowego słownika[3]

Słownikowe algorytmy kodowania polegają na kolejnym czytaniu sekwencji danych wejściowych i przeszukiwaniu słownika w dokładnie tej samej sekwencji danych. W przypadku zakończenia przeglądu sukcesem, indeks właściwej pozycji słownika staje się reprezentacją wyjściową przy czym im dłuższa jest sekwencja tym efektywniejszą jest kompresja. W kodowaniu słownikowym można także zastosować statyczny model słownika. Jest on skuteczniejszy, gdyż zna z wyprzedzeniem pojawiające się ciągi danych. Problemy związane z koniecznością dopisania do zbioru danych skompresowanych słownika użytego przy kodowaniu, są identyczne jak w metodach statycznych. Są stosowane w praktyce, ale jedynie do określonych przypadków kodowania w miarę stacjonarnych zbiorów danych o charakterystyce znanej a priori. Algorytmy przeznaczone do ogólnych zastosowań są w zdecydowanej większości adaptacyjne. W statycznym (entropijnym) kodowaniu pojedynczy symbol zastępowany jest ciągiem bitów o zmiennej długości. W przypadku kodowania słownikowa koncepcja jest odwrotna. Ciąg symboli o zmiennej długości jest zastępowany sekwencją kodową - indeksem w tablicy słownika. Ogólnie rzecz biorąc model ten polega na budowaniu słownika na podstawie znaków alfabetu, bądź to uzupełniany o słowa oparte na analizie danych pochodzących ze zbiorów kompresji. Następnie słownik ten jest zapisywany, bądź przesyłany do dekodera.[4]

Algorytm[4]:

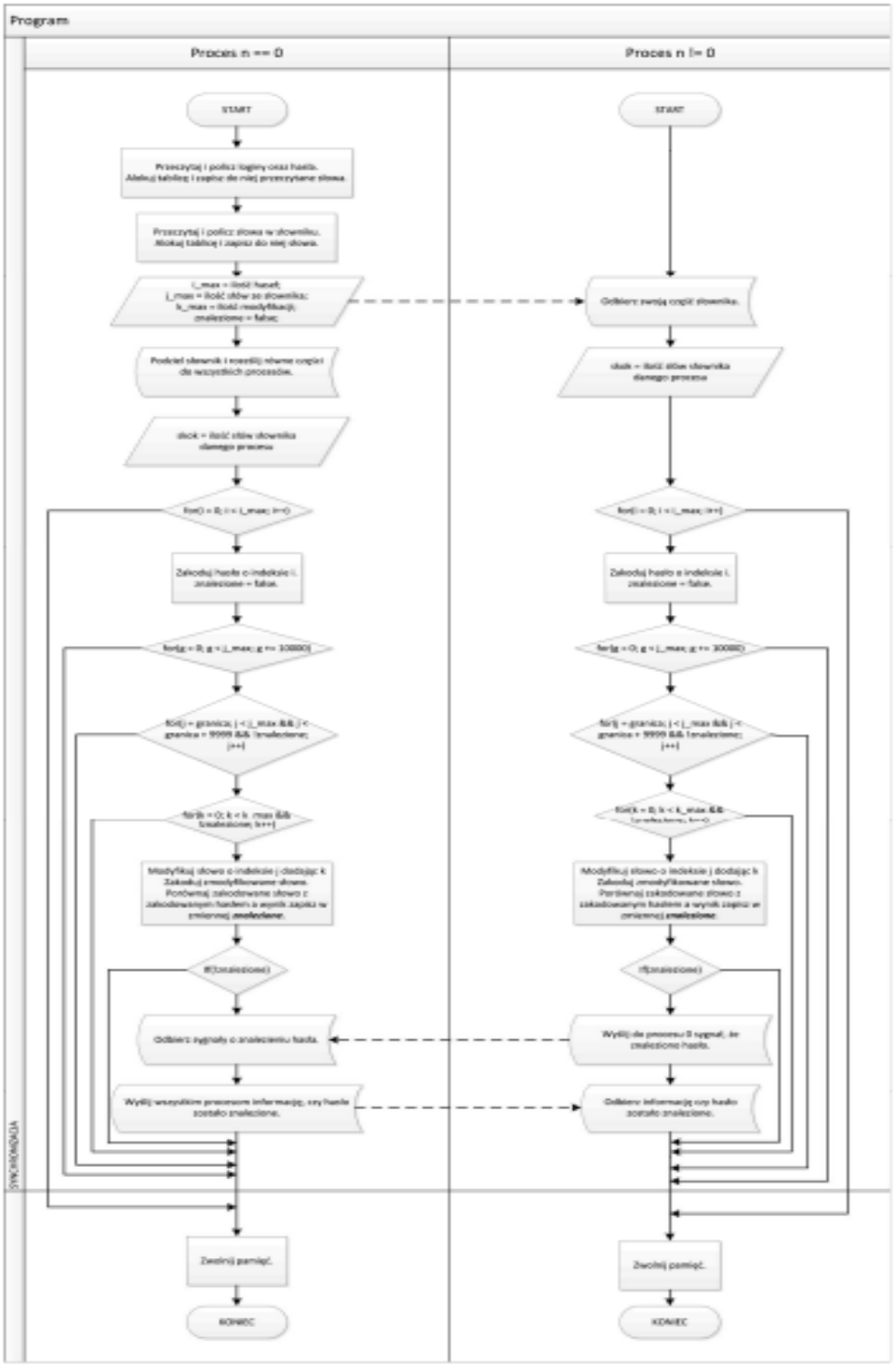
1. Inicjacja słownika - pierwsza pozycja jako NULL.
2. Przeszukaj aktualny słownik i ustal najdłuższy łańcuch z kolejno czytanych symboli zgodny ze słownikiem.
3. Utwórz słowo kodowe najdłuższego łańcucha: indeks + kolejny symbol.
4. Dopisz nową frazę do słownika (z pkt.3).
5. Powtarzaj od pkt.2 aż do zakodowania wszystkich symboli na wejściu.

Algorytm łamania haseł metodą słownikową jest techniką używaną do siłowego odgadywania haseł do systemów. Jego realizacja wymaga dużych nakładów obliczeniowych, dlatego też uzasadnione jest wykorzystanie do tego celu programowania równoległego. W artykule przedstawiono i porównano ze sobą, równoległe implementację tego algorytmu w trzech różnych środowiskach programowania równoległego, realizujące zrównoleglenie obliczeń na poziomie danych oraz środowisko procesorów kart graficznych[5].

Nie zawsze odzyskiwanie czy też testowanie siły haseł sprowadza się do łamania hashy. Wiele programów lub metod słownikowego odzyskiwania haseł wymaga podania wprost słownika haseł, które chcemy po kolei sprawdzić. Przykładem mogą być tu ataki online, czyli takie, które próbują odzyskać hasło na podstawie przeprowadzania kolejnych prób logowania do docelowego systemu/usługi.[3]

Rys. 1.2.1 Schemat blokowy algorytmu słownikowego.

2. Przykład innego rozwiązania.



Rys. 2.1Diagram aktywności dla implementacji.

Słownik jest dzielony na równe części przydzielane wszystkim procesom. Proces 0 czyta z pliku słownik oraz hasła do złamania. Najpierw je liczy, po czym alokuje potrzebną pamięć i do niej zapisuje zawartość plików. Potem oblicza ile słów przypada na jeden proces i wysyła do wszystkich potrzebne informacje. Na koniec dokonywany jest transfer słownika, aby każdy proces posiadał tylko tą jego cześć, na której ma operować. Pętla badająca kolejne słowa jest podzielona na mniejsze pętle po 10 tys. przejść. Zabieg ten ma na celu przerwanie pracy innych procesów po znalezieniu hasła przez jeden z nich. Aby nie tracić czasu na synchronizację wszystkich procesów w tym samym czasie, wysyłane są komunikaty nieblokujące. Procesem decydującym o zakończeniu poszukiwań jest proces 0. Każdy inny proces w przypadku złamania hasła wysyła do niego komunikat zgłaszający znalezienie hasła, po czym przesyła znalezione hasło. Proces 0 w przypadku otrzymania komunikatu o złamaniu hasła przez inny proces odbiera także komunikat zawierający złamane hasło. Jeśli proces 0 „wie", że hasło zostało złamane, wysyła taką informacje do wszystkich pozostałych procesów. Tylko proces 0 wyświetla informacje o tym, jakie są złamane hasła. Steruje całą komunikacją oraz zbiera informacje od wszystkich procesów. Ilość komunikacji w przypadku kilku rdzeni (procesorów) nie jest jednak na tyle duża, by wymagała osobnego procesu do wykonywania tego zadania, więc proces 0 wykonuje poszukiwania, tak jak i wszystkie inne procesy. Każdy proces co 10 tys. haseł sprawdza, czy nie został do niego wysłany komunikat informujący o złamaniu hasła przez inny proces. Jeśli otrzyma taki komunikat przerywa pracę. Diagram aktywności dla implementacji w środowisku MPI przedstawia rys. 1.

Rys. 2.2 Opis przedstawionego rozwiązania.

LITERATURA

[1] Algorytm brute force

http://mumin.pl/Skrypt_A_do_Z/lekcje/l06.html

[2] Działanie algorytmu brute force

http://prac.im.pwr.wroc.pl/~szwabin/assets/algo/lectures/5.pdf

[3] Opis ataków słownikowych.

https://sekurak.pl/crackstation-udostepnia-ogromny-slownik-hasel/

[4] Kodowanie słownikowe

https://www.ire.pw.edu.pl/~rois/dydaktyka/skrypt/5.pdf

[5] Działanie metody siłowej

http://neo.dmcs.p.lodz.pl/ptda/wyklady/wyklad_5.pdf

[6] Równoległe łamanie haseł

http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BGPK-3546-3483

[7] Schemat blokowy algorytmu słownikowego

https://slideplayer.pl/slide/11339059/

[8] Równoległe łamanie haseł - przykład

https://repozytorium.biblos.pk.edu.pl/redo/resources/32576/file/suwFiles/PlazekJ_RownoległeLamanie.pdf

[9] Brute force/metoda słownikowa

https://docplayer.pl/8991975-Akd-metody-slownikowe.html